LU TP 97-2
March 6, 1997

A Potts Neuron Approach to Communication Routing

Jari Häkkinen[1], Martin Lagerholm[2], Carsten Peterson[3] and Bo Söderberg[4]
Complex Systems Group, Department of Theoretical Physics
University of Lund, Sölvegatan 14A, SE-223 62 Lund, Sweden

Submitted to *Neural Computation*

Abstract:

A feedback neural network approach to communication routing problems is developed with emphasis on *Multiple Shortest Path* problems, with several requests for transmissions between distinct start- and endnodes. The basic ingredients are a set of Potts neurons for each request, with interactions designed to minimize path lengths and to prevent overloading of network arcs. The topological nature of the problem is conveniently handled using a propagator matrix approach. Although the constraints are global, the algorithmic steps are based entirely on local information, facilitating distributed implementations. In the polynomially solvable single-request case the approach reduces to a fuzzy version of the *Bellman-Ford* algorithm. The approach is evaluated for synthetic problems of varying sizes and load levels, by comparing with exact solutions from a branch-and-bound method. With very few exceptions, the Potts approach gives legal solutions of very high quality. The computational demand scales merely as the product of the numbers of requests, nodes, and arcs.

[1]jari@thep.lu.se
[2]martin@thep.lu.se
[3]carsten@thep.lu.se
[4]bs@thep.lu.se

Introduction

Communication routing resource allocation problems are becoming increasingly relevant given the upsurge in demand of internet and other telecommunication services. One such problem amounts to assigning arcs (links) in a connected network to requests from start- to endnodes, given capacity constraints on the links, such that a total additive cost (path-length) is minimized. For a review of notation and existing routing techniques, see e.g. ref. [1]. A relatively simple routing problem, with only one request at a time, is the Shortest Path Problem (**SPP**), which can be solved exactly in polynomial time using e.g. the Bellman Ford (**BF)** algorithm [1]. The Multiple Shortest Path Problem (**MSPP**), where links are to be allocated simultaneously to several requests, is more difficult. There exists, to our knowledge, no method that yields exact solutions to this problem in polynomial time.

In this paper we address the MSPP using feedback Potts neural networks, which have proven to be powerful in other resource allocation problems, with [2] or without [3] a non-trivial topology. For each request we assign a Potts network, with units encoding which links to be used by that request. Appropriate energy terms are constructed in terms of the Potts neurons to minimize total path lengths and to ensure that capacity constraints are not violated. Mean field (**MF**) equations are iterated using annealing to minimize the total energy. In contrast to earlier usage of Potts encoding and MF annealing [4, 3, 2] where global objective functions are minimized, here each node minimizes its own local energy.

For the case of a *single* request, the Potts MF approach reduces in the zero temperature limit to the BF algorithm; hence our approach contains this standard algorithm as a special case.

For each request the Potts MF network [4] defines a "fuzzy" *spanning tree*[5], rooted at the endnode. In order to project out the (fuzzy) path from the startnode in this subgraph, and to keep track of the paths in general, we utilize a propagator matrix formalism following ref. [2]. The computation of the propagator requires matrix inversion; fortunately this can be done using an iterative procedure with low computational cost.

As in a previously considered airline crew scheduling problem [2], proper preprocessing is employed to identify independent subproblems, in order to reduce the problem complexity.

Despite the existence of global constraints, the implementation of the approach is truly local – when updating the MF equations for a particular node, only information residing at neighbouring nodes is needed.

The approach is gauged by an exact branch-and-bound (**BB**) algorithm on a set of synthetic but challenging test problems, showing an excellent performance of the Potts MF approach, with a CPU consumption per request scaling merely like NN_L, where N is the number of nodes and N_L the number of links in the network. The method is also very robust with respect to parameters. Due to the excessive demand for CPU resources by the reference BB method, our comparisons are limited to fairly low problem sizes. However, there are no indications that the Potts method be less efficient for larger problems.

[5]A subgraph connecting all nodes without loops.

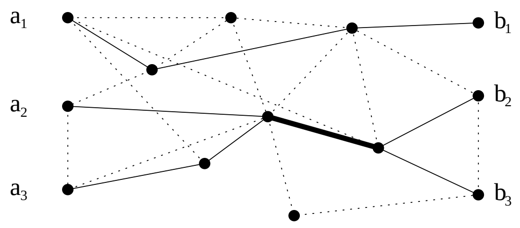

Figure 1: Example of a solution to a 3-request problem. Dotted lines represent unused links, and full lines links used by the requests.

The Multiple Shortest Path Problem

A Multiple Shortest Path Problem (MSPP) is defined by specifying the following:

- A connected network of N nodes and N_L links (arcs).
- For each arc ij, a cost (arc-length) d_{ij} and a capacity C_{ij}.
- A set of N_R transmission requests r, each defined by a startnode a_r and an endnode b_r.

The task is then to assign to each request a connected loop-free path of arcs from the startnode to the endnode. This is to be done such that the total length of those paths is minimized, without the load on any arc exceeding its capacity, with the load defined as the number of requests sharing it. A 3-request problem example is given in fig. 1.

All problems of this kind are not solvable; a reliable algorithm should be able to recognize and signal a failure, to enable proper measures to be taken.

The Bellman-Ford Algorithm in the Mean Field Language

Prior to dealing with the MSPP, we revisit the simpler SPP and demonstrate how the BF algorithm can be recast in a Potts MF language. This formulation will then be the starting point for designing a Potts MF approach to MSPP.

In the SPP there is only a single request, from a to b, and the capacity constraints are irrelevant. The task simply is to find the shortest path from a to b. In the BF algorithm [1] this is achieved by minimizing the path-lengths D_i from every node i to b, by iterating

$$D_i \rightarrow \min_j (d_{ij} + D_j), \quad i \neq b. \tag{1}$$

and keeping track of the involved links ij. D_b is fixed to zero by definition. The resulting solution defines a spanning tree rooted at b. In particular, D_a is determined, and the minimal path from a to b is easily extracted from the spanning tree.

If no link exists from node i to j, d_{ij} could formally be defined to be infinite; in practice it is more convenient to restrict j in eq. (1) to the actual *neighbours* of i, reachable via an arc from i.

Eq. (1) can be rewritten as

$$D_i = \sum_j v_{ij} E_{ij} \equiv \sum_j v_{ij}(d_{ij} + D_j) \tag{2}$$

in terms of a *Potts spin* $\mathbf{v}_i$ for every node $i \neq b$, with components v_{ij} taking the value 1 for the j with the smallest *local energy* E_{ij}, and 0 for the others (winner-takes-all). Note the distinct philosophy here: each node i minimizes its own local energy $D_i = \min_j E_{ij}$, rather than all nodes striving to minimize some global objective function.

A *mean field* (MF) version of eq. (2) is obtained by using for $\mathbf{v}_i$ its thermal average in the MF approximation, defined by

$$v_{ij} = \frac{e^{-E_{ij}/T}}{\sum_k e^{-E_{ik}/T}} \tag{3}$$

where j and k are neighbours of i, and T is an artificial temperature. Note that each *Potts MF neuron* $\mathbf{v}_i$ obeys the normalization condition

$$\sum_j v_{ij} = 1 \tag{4}$$

allowing for a probabilistic interpretation of the components.

At a non-zero temperature, iteration of eqs. (2,3) can be viewed as a fuzzy implementation of the BF algorithm, while in the $T \to 0$ limit the neurons are forced *on-shell*, i.e. $v_{ij} \to 1$ (for the minimizing j) or 0 (for the rest), and proper BF is recovered.

Given this obvious neural recast of the BF algorithm in terms of Potts neurons, it is somewhat surprising that non-exact neural approaches based on Ising spins have been advocated in the literature [7].

In addition, the MF Potts algorithm considered here exhibits a close electrostatic analogy in terms of Kirchhoff's laws on a graph [8].

The Potts Mean Field Approach to MSPP

The Potts MF formulation of the Bellman-Ford algorithm for SPP (eqs. (2,3)) is a suitable starting point for approaching MSPP.

We will stick with the philosophy inherited from BF of focusing on independent *local energies*, in contrast to what has become standard when using feedback neural networks for resource allocation problems. This represents a novel strategy.

Thus, we introduce a separate Potts system, $\{v^r_{ij}\}$, for each request r, with basic local energies E_{ij} as before representing distances to the endnode b_r. In addition, we will need energy terms E^{load}_{ij} for the load-constraint, to be discussed later; this introduces an interaction between the Potts systems.

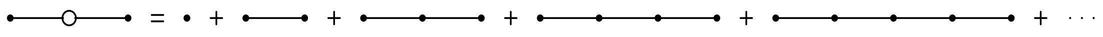

Figure 2: Expansion of the propagator P^r_{ij} in terms of v^r_{kl} (—) and intermediate nodes (•).

This formulation introduces the possibility of undesired loop formation, since forming a loop might induce less energy penalty than violating a load constraint. As will be discussed below such loops can be suppressed by a suitable penalty term, E^{loop}_{ij}, and by adding a possibility for each proper node to connect, via an artificial *escape link*, to an artificial *escape node*, for each request connecting directly to the endnode. This enables a "give-up" state for an unresolvable situation, signaled by some path containing the escape node. The cost for "giving up" must be larger than that for any legal path. Therefore the length of each escape link is set to the sum of the lengths of the proper links, while the corresponding capacity is chosen large enough to be able to host all the requests.

In order to terminate the path for a request r, its endnode must be a *sink* for the corresponding Potts system. Consequently, there will be no Potts neuron $\mathbf{v}^r_{b_r}$ associated with it.

In order to construct appropriate penalty terms, a propagator matrix will be used. This technique has proved to be a powerful tool in neural optimization for problems with a non-trivial topology [2]. In particular, it will be crucial for extracting properties of the fuzzy paths defined by the MF approach at finite T.

Path Extraction and the Propagator

The normalization condition (eq. (4)) ensures that for each request there is precisely one continuation for each node, although for $T \neq 0$ it is fuzzily distributed over the available neighbours. On shell, the path from start- to endnode is trivial to extract – one follows the $v^r_{ij} = 1$ path starting from the startnode. However, for $T \neq 0$ a more refined path extraction mechanism is needed. This is provided by a *propagator matrix* [2] $\mathbf{P}^r$ for each request r, defined by:

$$P^r_{ij} = \left[\left(\mathbf{1} - \mathbf{v}^r \right)^{-1} \right]_{ij} = \delta_{ij} + v^r_{ij} + \sum_k v^r_{ik} v^r_{kj} + \sum_{kl} v^r_{ik} v^r_{kl} v^r_{lj} + \ldots \tag{5}$$

For a graphical representation, see fig. 2. On shell, it is easy to see that P^r_{ij} can be interpreted as the *number of paths* from i to j defined by the Potts neurons associated with r; similarly, the elements of the matrix square $(P^r)^2_{ij}$ are related to the number of arcs used in those paths[6].

Off shell, these interpretations are still valid in a probabilistic sense. A probabilistic measure of how much node i participates in the path $a_r \to b_r$ is given by $P^r_{a_r i} P^r_{i b_r} \equiv P^r_{a_r i}$, the simplification being due to the identity $P^r_{i b_r} = 1$, following from eq. (4). This provides us with a natural path extractor; in particular, we have on shell (in the absence of loops)

$$P^r_{a_r i} = \begin{cases} 1, & \text{if node } i \text{ appears in the path } a_r \to b_r \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

[6]More precisely, the number of arcs is given by $P^2 - P$.

Load and Loop Control

Armed with the propagator formalism, we proceed to set up the penalty terms, to be added to the energies E_{ij} corresponding to eq. (2).

In order to efficiently avoid complications from self-interactions in the local energies, independently of T, we follow ref. [5], and define the penalty terms based on analyzing the result of setting one component v_{ij}^r of the neuron $\mathbf{v}_i^r$ at a time to one, with the other components set to zero, as compared to a reference state with all components set to zero.

The total load L_{ij} on a link ij is the sum of contributions from the different requests,

$$L_{ij} = \sum_r L_{ij}^r. \tag{7}$$

The contribution from request r can be expressed as

$$L_{ij}^r = \frac{P_{a_r i}^r}{P_{ii}^r} v_{ij}^r \tag{8}$$

where P_{ii}^r corrects for possible improper normalization due to soft loop-contributions.

The load constraints, $L_{ij} \leq C_{ij}$, define a set of inequality constraints. In the realm of feedback neural networks, such constraints have been successfully handled by means of step-functions [5]. For a given request r, the available capacity of the link ij is given by

$$X \equiv C_{ij} - L_{ij} + L_{ij}^r\ , \tag{9}$$

in terms of which an overloading penalty can be defined as

$$E_{ij}^{\text{load}} = (1 - X)\,\Theta\,(1 - X) + (X)\,\Theta\,(-X) \tag{10}$$

where $\Theta()$ is the Heaviside step-function. Eq. (10) expresses the additional overloading of the link, if it were to be used by r.

The amount of loops introduced by connecting $i \rightarrow j$ can be expressed as

$$Y \equiv \frac{P_{ji}^r}{P_{ii}^r} \tag{11}$$

A suitable loop suppression term is then given by

$$E_{ij}^{\text{loop}} = \frac{Y}{1 - Y}. \tag{12}$$

The generalization of the local energy in eq. (2) to the multiple request case now reads, for a particular request r,

$$E_{ij} = d_{ij} + D_j^r + \alpha E_{ij}^{\text{load}} + \gamma E_{ij}^{\text{loop}} \tag{13}$$

with the added terms based on eqs. (10,12). The resulting algorithm allows for a wide range of choices of the coefficients α and γ without severely changing the performance.

Updating Equations

All neurons are repeatedly updated, with a slow annealing in T. For each request r and each node i, the corresponding neuron $\mathbf{v}_i^r$ is updated according to

$$v_{ij}^r = \frac{e^{-E_{ij}/T}}{\sum_k e^{-E_{ik}/T}}. \tag{14}$$

with E_{ij} given by eq. (13). The corresponding length D_i^r from node i to the endnode is then updated, in the BF spirit, as

$$D_i^r \rightarrow \sum_j v_{ij}^r E_{ij}. \tag{15}$$

The corresponding update of the propagator could in principle be done using an exact incremental matrix inversion scheme like Sherman-Morrison [6]. We prefer, though, to let local changes propagate through the network, in analogy to the update of D_i^r. Thus, only the i'th row of $\mathbf{P}^r$ is updated:

$$P_{im}^r \rightarrow \delta_{im} + \sum_j v_{ij}^r P_{jm}^r \ , \ \text{ for all } m \tag{16}$$

This gives a convergence towards the exact inverse, which turns out to be good enough. The advantage of this method is twofold: it is faster, and all information needed is local to the relevant node i and its neighbours j (assuming each node to keep track of its own row of $\mathbf{P}^r$).

Details of the algorithmic steps and the initialization can be found in Appendix A.

Test Problems and Explorations

In order to test the Potts MF method we have generated a set of synthetic problems. In doing so we face limitations given by the ability of the exact reference method (BB) to provide solutions, due to limited computing power. For this reason our comparisons are restricted to relatively small systems.

With these limitations in mind, we have tried to choose the test problems as difficult as possible. The most important parameters governing the difficulty of a problem, apart from network size and connectivity, are the number of requests and the average link capacity. In cases where all the links are able to host all the requests, $C_{ij} \geq N_R$, the problem is separable, and can be solved by running an independent BF algorithm for each request. Therefore we choose to work with very tight link bounds.

For each problem, we generate a random connected network, where every node has at least one path to all other nodes. To that end, all nodes are first connected in a random spanning tree. Additional links (creating loops) are then randomly placed. Every link is given a random capacity and a random length. Finally a specified number of random requests are generated, in terms of start- and endnodes. An example of a generated test problem is shown in fig 3.

This procedure does not automatically yield a solvable problem, where all requests can be fulfilled simultaneously without violating any constraint. In principle, solvability could be built into the

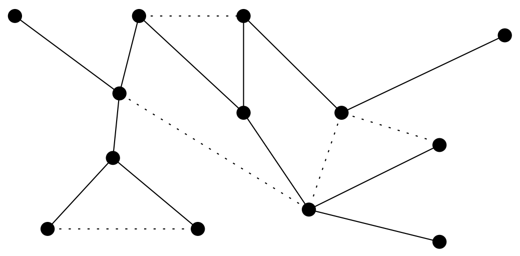

Figure 3: An example network with 13 nodes and 17 links. The initial spanning tree are the solid lines whereas the dotted lines represent the links creating loops.

problem-generator. Here we adopt another strategy, by creating random problems and attempting to solve them exactly; those with the solution not found by BB within a certain amount of CPU time are disregarded. In principle, this method could introduce a bias towards simple problems. However, for the problem sizes considered only a tiny fraction of the problem candidates are "timed-out", except for the largest problems (last row in Table 3), where about 1/3 are disregarded.

Prior to attacking a problem, a decomposition into independent sub-problems is attempted, to reduce complexity. The required computer time for a decomposed problem is dominated by the largest sub-problem. All sub-problems are solved in the explorations described below. The CPU demand of the MF approach scales like $N_R N N_L$.

The performance of the MF Potts approach is probed my measuring the excess path length as compared to the BB result,

$$\Delta = \frac{D_{\rm MF} - D_{\rm BB}}{D_{\rm BB}}. \tag{17}$$

where $D_{\rm MF}$ and $D_{\rm BB}$ are the total path lengths resulting from the two algorithms. In Table 1 we show characteristics of the generated test problems, together with performance measures for the Potts MF algorithm. As a measure of the complexity of a problem we use the entropy, S, defined as the logarithm of the total number of possible configurations, disregarding load constraints.

Nodes	Links	Requests	$< S >$	% legal	$< \Delta >$	<CPU-time>
5	10	5	14	100.0	0.003	0.1
5	10	10	28	99.9	0.002	0.2
10	15	10	28	100.0	0.004	0.4
10	20	10	48	100.0	0.003	0.5
15	20	10	27	99.8	0.03	0.5
15	20	15	40	99.9	0.06	0.7

Table 1: Results averaged over 1000 problems of each size. The entries "% legal" and <CPU time> refer to the MF Potts approach. The time is given in seconds using DEC Alpha 2000. Typical times for the BB method are around 600 seconds.

Table 1 indicates an excellent performance of the MF Potts approach, with respect to giving rise to legal solutions with good quality, with a very modest computational demand.

Summary and Outlook

We have developed a Potts MF neural network algorithm for finding approximate solutions to the Multiple Shortest Path Problem. The starting point is a Potts MF recast of the exact Bellman-Ford algorithm for the simpler single Shortest Path Problem. This approach is then extended to the Multiple Shortest Path Problem by utilizing several Potts networks, one for each request. Complications of topological nature are successfully handled by means of a convenient propagator approach, which is crucial for the following issues:

- The MF approach yields at $T \neq 0$ fuzzy spanning trees, from which the propagator is used to extract fuzzy paths.
- The load-constraints are handled by energy terms involving the propagator.
- Loops are suppressed by energy terms, also based on the propagator.

In addition, an auxiliary link to an escape node is introduced for each proper node, opening up escape paths for unresolvable situations.

The method is local in that only information available from neighbouring nodes is required for the updates. This attractive feature, inherited from the BF algorithm, facilitates a distributed implementation.

The computational demand of the method is modest. The CPU time scales as $N_R N N_L$. With fixed connectivity this corresponds to $N_R N^2$, whereas for the worst case of full connectivity it yields $N_R N^3$.

The performance of the algorithm is tested on a set of synthetic challenging problems, by comparing to exact results from a branch-and-bound method, for various problem sizes. The comparison shows that the Potts MF approach with very few exceptions yields very good approximate solutions.

The method is presently being generalized to other routing problems.

Appendix A. The Potts MF Algorithm

Initialization

The initial temperature T_0 is first set to $T_0 = 50$. If the saturation Σ,

$$\Sigma \equiv \frac{1}{N_R(N-1)} \sum_{i \neq b_r} \mathbf{v}_i^2, \tag{A1}$$

has changed more than 10% after all neurons have been updated once, then the system is reinitialized with $T_0 \to 2T_0$.

For all nodes (except the end- and escape nodes), the corresponding Potts neurons are initialized in accordance with the high temperature limit, i.e.

$$v_{i,j}^r = 1/n_i \tag{A2}$$

for all n_i neighbours j (including the escape node) of i . P_{ij}^r and D_i^r are initialized consistently with eq. (A2).

Iteration

```
Until $T \leq T_f$ or $\Sigma \geq \Sigma_f$ do:

  • For every request $r$ do:
      1. For every node $i$ except $b_r$ and the escape node:
         (a) Update $\mathbf{v}_i^r$ (eqs. (3,13)).
         (b) Update $D_i^r$ (eq. (15)).
         (c) Update $\mathbf{P}_i^r$ (eq. (16)).
      2. Update $L_{ij}$.
  • Decrease the temperature: $T = kT$.
```

Parameters used are: $k = 0.90$, T_f=0.0001, Σ_f=0.99999. For the energy coefficients in eq. (13), we have consistently used $\alpha = 1$ and $\gamma = 5$.

References

[1] D. Bertsekas and R. Gallager, *Data Networks*, Prentice-Hall, New Jersey (1987).

[2] M. Lagerholm, C. Peterson and B. Söderberg, "Airline Crew Scheduling with Potts Neurons", *LU TP 96-6* (to appear in *Neural Computation*).

[3] L. Gislén, C. Peterson and B. Söderberg, "Complex Scheduling with Potts Neural Networks", *Neural Computation* **4**, 805 (1992).

[4] C. Peterson and B. Söderberg, "A New Method for Mapping Optimization Problems onto Neural Networks", *International Journal of Neural Systems* **1**, 3 (1989).

[5] M. Ohlsson, C. Peterson and B. Söderberg, "Neural Networks for Optimization Problems with Inequality Constraints – the Knapsack Problem", *Neural Computation* **5**, 331 (1993).

[6] See e.g. W.P. Press, B.P Flannery, S.A. Teukolsky and W.T. Vettering, *Numerical Recipes, The Art of Scientific Computing*, Cambridge University Press, Cambridge (1986).

[7] S.C.A. Thomopolous, L. Zhang and C.D. Wann, "Neural Network Implementation of the Shortest Path Algorithm for Traffic Routing in Communication Networks", *Proceedings of the IEEE International Joint Conference on Neural Networks*, Singapore, 2693 (1991).

[8] J. Häkkinen, M. Lagerholm, C. Peterson and B. Söderberg, in preparation.